BM1880 Darknet Yolov2/3 model 转 bmodel 说明

版本: V1.0.4

发布时间：2019.09.27

Revision History

Revision Number	Author	Date	Description
1.0	Hongjun.Chu	2019.06.10	Initial Draft
1.0.1	Hongjun.Chu	2019.07.01	Attach new version calibration/tuning tool
1.0.2	Hongjun.Chu	2019.07.05	添加了对模型精度有影响的操作注意事项
1.0.3	Hongjun.Chu	2019.07.12	修改了 darknet cfg 文件参数说明，增加 fp32/int8 model 的测试方法。
1.0.4	Hongjun.Chu	2019.09.27	文档内所用的 tool 的路径的调整，字体的调整。

目录

目前我们提供了 Yolov2/3 bmodel 在 darknet 框架下作推理的 demo 实例(https://github.com/BM1880-BIRD/bm1880-ai-demo-program/tree/master/darknet-yolov2-object-classification-v2)，下面说明一下如何将 Darknet 训练出来的 yolov2/3 model 转为 bmodel 以方便开发者做基于 yolo 网络的客制化。

1 Darknet model（cfg 和 weights）如何转 bmodel.

目前大部分的 yolo 网络的训练过程都是在 Darknet 框架内完成的，但目前 BM1880 只能基于 Caffe model 做 INT8 模型(bmodel)的转换. 所以有了 Darknet model 第一件事情就是 Darknet 转 Caffe model. 下面步骤说明如何完成转换。

1.1 相关 Tool 的安装

Tool 的安装，请在 Edge-Development-Toolchain(https://github.com/BM1880-BIRD/Edge-Development-Toolchain)的 docker 环境中安装运行以下的流程。会省掉您很多配置环境的时间。

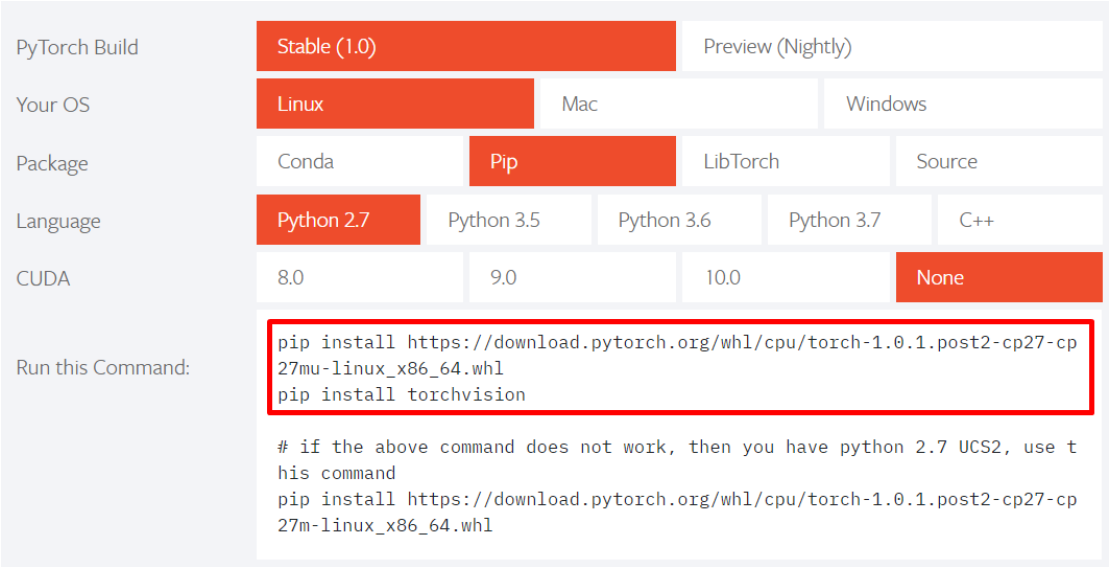
笔者转换主要用到了下面两个 git hub 上的开源 Tool .

https://github.com/ChenYingpeng/caffe-yolov3
https://github.com/marvis/pytorch-caffe-darknet-convert

主要用到了前者中 caffe-yolov3/yolov3_darknet2caffe.py 做 yolov3 到 caffe model 的转换；用后者内的 darknet2caffe.py 做 yolov2 到 caffe model 的转换.

这两个 python script 的成功运行需要先安装 Pytorch 和 Caffe . Pytorch 笔者用的是 pip 安装方式：请参考：https://pytorch.org/ 按需求选择安装即可。

笔者的安装选择：



Caffe 的安装这里不详细描述(Caffe 需加入对 upsample 以及 reorg 两个 layer 支持，网上可以自行查找)。对于 Caffe 的安装不熟悉的用户，可以直接用我们提的 Edge development Toolchain 中的 callfe(https://github.com/BM1880-BIRD/Edge-Development-Toolchain).

由于 darknet2caffe.py 目前只支持 tiny yolo model 转 Caffe，少了对 reorg 和多参数 route layer 的转换支持。这里笔都加了如下的 patch 以支持这两个 layer.

darknet2caffe.patch

对于 yolov3 model， BM1880 的 calibration tool 和最终的 run time lib 会吃的 upsample layer 的参数是 size (而不是 Scale）.所以需要加上如下的 patch.

yolov3_darknet2caffe.py.patch

1.2 Darknet model（cfg 和 weights）如何转 bmodel.

1.2.1 在没有自己 Caffe 的 prototxt 情况下的模型转换

这种转换方式的方便之处是，不用再另外写 Caffe 的 prototxt ，只需要有 darknet 下的 cfg 和 weights 文件即可。

笔者将 caffe-yolov3/yolov3_darknet2caffe.py 拷贝到 pytorch-caffe-darknet-convert 目录下以方便环境变量的设置。

yolov3 和 yolov2 的 model 转 Caffe 分别执行如下命令，最终会生成 Caffe 的 prototxt 和 caffemodel.

- Yolov3:

python yolov3_darknet2caffe.py yolov3.cfg yolov3.weights yolov3.prototxt yolov3.caffemodel

- Yolov2:

python darknet2caffe.py yolov2.cfg yolov2.weights yolov2.prototxt yolov2.caffemodel

最终生成的 caffemodel，只需要保留到卷积 layer 作为 output，如下所示。


yolov2.prototxt


yolov3.prototxt

特别注意：

对于转完的 yolov3.prototxt，里面有单 input 的 concat layer，这个 layer 要拿掉。否则会影响到 INT8 的推理。实例如下：


yolov3 - 修改前.prototxt


yolov3 - 修改后.prototxt

1.2.2 在自己生成 Caffe 的 prototxt 情况下的模型转换

如果有自己实现的 Caffe model (prototxt), 可以用如下的 python script 完成 caffe model 的生成。将 yolov2.prototxt 和 yolov2.weights 放到相同的目录。

Yolov2:
python convert_weights_to_caffemodel_yolo2.py

convert_weights_to_caffemodel-yolov2.py

Yolov3:
python convert_weights_to_caffemodel_yolo3.py

convert_weights_to_caffemodel_yolov3.py

1.3 模型量化生成 bmodel

用 calibration tool 作量化并转换生成 bmodel，请参考 Edge-Development-Toolchain 内 calibration 的 Resnet 网络量化实例。具体步骤可以参考 https://github.com/BM1880-BIRD/Edge-Development-Toolchain/blob/master/calibration_tool/Calibration-Tool-Guide.pdf 第 4 章。

需要注意的是 prototxt input layer 的写法，笔者转的是原生 model ，这里要做 data 归一化 (0.00392156862 = 1/255)(如下示例) ，客制化 model 需要与原 model train 的时候的 data 处理保持一致。

```
layer {
  name: 'data'
  type: 'Python'
  top: 'data'
  python_param {
    module: 'general_data_layer'
```

```
    layer: 'DataLayer'
    param_str: "{'data_list': /home/chjtest/compiler/Release/input.txt, 'color_format': rgb, 'h': 608, 'w': 608,'scale': 0.00392156862}"
  }
}
```

1.4 在 caffe 跑测试程序测试转完的 model 精度

目前在 Edge-Development-Toolchain sample (https://github.com/BM1880-BIRD/Edge-Development-Toolchain/tree/master/samples/detection) 中，有 yolo 的推理实例，可参考做离线的精度测试。步骤如下：

1. 跑 fp32 的 caffe model 的结果与 darknet 中的结果进行比较，看是否在 darknet2caffe 的时候有精度掉。
2. 跑 int8 的结果与跑 fp32 的 model 的结果做比较，看是否在 calibration int8 的时候有精度掉。

经验分享：

1. 根据目前客户的反馈，上面 1 中 darknet2caffe 过程，精度一般都不会掉。
2. 转完的 int8 会有精度损失。这个时候视情况首先检查相应的参考是否设置对，几个点需要检查：
   1) 转 caffe model 后，prototxt 中是否有手动拿掉单 input concat；
   2) Calibration 的时候，input data 的前处理参数是否有设置对，比如下面红框里的设置。

```
layer {
  name: 'data'
  type: 'Python'
  top: 'data'
  python_param {
    module: 'general_data_layer'
```

```
7.      layer: 'DataLayer'
8.      param_str: "{'data_list': /home/chjtest/compiler/Release/input.txt, 'col
    or_format': rgb, 'h': 608, 'w': 608,'scale': 0.00392156862}"
9.    }
10. }
```

3) 在做离线测试的时候，FP32 的 prototxt input layer 是否有写对。

```
4)  layer {
5)    name: "data"
6)    type: "Input"
7)    top: "data"
8)    input_param {
9)      shape: { dim: 1 dim: 3 dim: 608 dim: 608 }
10)     enable_quantize: 1
11)   }
12) }
```

跑完这一步，如果 int8 的精度也满足了，就可以接下来第 2 章的内容。否则，需要转到第 3 章节做精度调优。

2 将 bmodel 导入到 darknet 中利用 BM1880 完成推理

目前有提供 Yolo model 在 Darknet 下面的推理 demo.

https://github.com/BM1880-BIRD/bm1880-ai-demo-program/tree/master/darknet-yolov2-object-classification-v2

2.1 将自己的 bmodel 换到下的目录

darknet-yolov2-object-classification-v2/darknet/models/bmnet

2.2 更改 darknet 运行的 cfg 文件

- Yolov2 model:

  darknet/cfg/bmnet_yolov2.cfg. 下面中为 yolov2 608 实例，红色加粗字体需要

根据自

己的模型进行修改。参数如何设置都有相关的注释说明。

```
[net]
# Testing
batch=1
subdivisions=1
#模型输入的宽高
width=608
height=608
channels=3
momentum=0.9
decay=0.0005
angle=0
saturation = 1.5
exposure = 1.5
hue=.1
learning_rate=0.001
burn_in=1000
max_batches = 500200
policy=steps
steps=400000,450000
scales=.1,.1

[bmnet]
#模型输出的宽高
out_h=19
out_w=19
# 模型输出的 channel 数，out_c=5*(4+1+classes), classes 是物体检测的种类数量。
out_c=425
#转完的 bmodel 路径
bmodel_file=models/bmnet/yolov2_1_3_608_608.bmodel

[region]
#anchors 要与模型训练时保持一致，注意顺序
anchors =  0.57273, 0.677385, 1.87446, 2.06253, 3.33843, 5.47434, 7.88282, 3.52778, 9.77052, 9.16828
bias_match=1
#检测物体种类数
classes=80
coords=4
num=5
```

```
softmax=1
jitter=.3
rescore=1

object_scale=5
noobject_scale=1
class_scale=1
coord_scale=1

absolute=1
thresh = .6
random=1
```

- Yolov3 model:

  darknet/cfg/bmnet_yolov3.cfg. 下面中为 yolov3 608 实例，红色加粗字体需要根据自己的模型进行修改。参数如何设置都有相关的注释说明。

```
[net]
batch=1
subdivisions=1
#模型 input 宽高
width=608
height=608
channels=3
momentum=0.9
decay=0.0005
angle=0
saturation = 1.5
exposure = 1.5
hue=.1

learning_rate=0.001
burn_in=1000
max_batches = 500200
policy=steps
steps=400000,450000
scales=.1,.1

[bmnet]
#out_w/h 模型最终输出的宽高，out_h=模型输入高/32 , out_w=模型输入宽/32
out_h=19
```

```
out_w=19
#out_c 是 bmnet 的输出的 channel 数。计算方法(19*19*255+38*38*255+76*76*255) /19*19,
#bmnet 的输出 = out_h*out*width*out_c
#255=3*(4+1+80), 4 是输出的 bbox 的 x/y/width/height，1 是 box 的置信度，80 是检测类
#别数，3 是共有 3 个尺度的框。
out_c=5355
#转完的 bmodel 路径。
bmodel_file=models/bmnet/yolov3_1_3_608_608.bmodel

[yolo]
#yolo layer 会从 bmnet 的输出中通过 input_offset 去抓对应位置的数据最终算出检测结果。
#第一个 yolo offset 设置为 0
input_offset=0
#第一个尺度框的大小。
input_h=19
input_w=19
#mask 如果用的 anchor 是 9 个就不用变。如果是 anchor 有变要重新设置.例：如果 anchor
数量是 15，这里就是 10,11,12,13,14. (总数分三组的最后一组,0 开始)
mask = 6,7,8
#anchor box 设置需要与模型 training 里，保持一置，但要注意设置的顺序。
anchors = 10,13, 16,30, 33,23, 30,61, 62,45, 59,119, 116,90, 156,198,
373,326
#Total 类别
classes=80
#anchor 数量
num=9
jitter=.3
ignore_thresh = .7
truth_thresh = 1
random=1

[route]
layers = -2

[yolo]
#第二个 yolo layer 的 offset , 19*19*255=92055
input_offset=92055
#第二个尺度框的大小
input_h=38
input_w=38
#mask 如果用的 anchor 是 9 个就不用变。如果是 anchor 有变要重新设置.例：如果 anchor
数量是 15，这里就是 5,6,7,8,9(总数分三组的最后二组,0 开始)
mask = 3,4,5
anchors = 10,13, 16,30, 33,23, 30,61, 62,45, 59,119, 116,90, 156,198,
```

```
373,326
classes=80
num=9
jitter=.3
ignore_thresh = .7
truth_thresh = 1
random=1

[route]
layers = -4

[yolo]
#第二个 yolo layer 的 offset , 19*19*255+38*38*255=460275
input_offset=460275
#第三个尺度框的大小
input_h=76
input_w=76
#mask 如果用的 anchor 是 9 个就不用变。如果是 anchor 有变要重新设置.例：如果 anchor 数量是 15，这里就是 0,1,2,3,4(总数分三组的最后三组,0 开始)
mask = 0,1,2
anchors = 10,13, 16,30, 33,23, 30,61, 62,45, 59,119, 116,90, 156,198, 373,326
classes=80
num=9
jitter=.3
ignore_thresh = .7
truth_thresh = 1
random=1
```

2.3 参考 Readme 编译并运行

目前我们同时支持 USB mode 和 SOC mode 两种方式。

3 INT8 模型精度的调优

以目前已发生的情况来看，在按如上的步骤转换成 INT8 yolo model 都会遇到不同情况的精度损失。精度的调优对于最终产品落地也是不可缺少的一个环节。

精度调整的工具可以使用 Edge-Development-Toolchain 中的 tuning tool(https://github.com/BM1880-BIRD/Edge-Development-Toolchain/tree/master/tuning_tool). 具体的方法可以参见如下文档。

https://github.com/BM1880-BIRD/Edge-Development-Toolchain/blob/master/tuning_tool/Auto-Tuning-Tool-Guide.pdf